Assignment1

February 20, 2025

1 GDS Assignment 1

Rasmus Dübeck Kristensen

1.1 Part 1

I formulated a regular expression to filter CPR numbers and included a number of testcases to check that the correct centuries were returned, which they were.

[33]:
```python
# Import regular expression module
import re

CPRdata = '''
123456-1234
1234561234
483493-1234
4834931234
324567-1234
3245671234
984321-1234
9843211234
'''

# Create regular expression
pattern = re.compile("^([0-9]{2})([0-9]{2})([0-9]{2})-?([0-9]{4})")

for data_line in CPRdata.split('\n'):

    match = pattern.match(data_line)

    if match: # If match is found, assign groups to variables
        DD = int(match.group(1))
        MM = int(match.group(2))
        YY = int(match.group(3))
        IIII = int(match.group(4))

# Function to return century based on CPR number
def return_century (IIII, YY):
    if IIII in range(1, 3999) and YY in range(00,99):
        return 1900
    elif IIII in range(4000, 4999) and YY in range(00, 36):
        return 2000
    elif IIII in range(4000, 4999) and YY in range(37, 99):
        return 1900
    elif IIII in range(5000, 8999) and YY in range(00, 57):
        return 2000
    elif IIII in range(5000, 8999) and YY in range(58, 99):
        return 1800
    elif IIII in range(9000, 9999) and YY in range(00, 36):
        return 2000
    elif IIII in range(9000, 9999) and YY in range(37, 99):
        return 1900
    else:
        return "Invalid CPR number"

# Testcases (succesful):
print(f"Person 1: 1234, 56: Born in century {return_century(1234, 56)}")
print(f"Person 2: 4222, 11: Born in century {return_century(4222, 11)}")
print(f"Person 3: 4222, 76: Born in century {return_century(4222, 76)}")
print(f"Person 4: 5009, 11: Born in century {return_century(5009, 11)}")
print(f"Person 5: 5009, 76: Born in century {return_century(5009, 76)}")
print(f"Person 6: 9001, 11: Born in century {return_century(9001, 11)}")
print(f"Person 7: 9001, 41: Born in century {return_century(9001, 41)}")
```

```
Person 1: 1234, 56: Born in century 1900
Person 2: 4222, 11: Born in century 2000
Person 3: 4222, 76: Born in century 1900
Person 4: 5009, 11: Born in century 2000
Person 5: 5009, 76: Born in century 1800
Person 6: 9001, 11: Born in century 2000
Person 7: 9001, 41: Born in century 1900
```

2 Part 2

1: I only considered the column "content" since I found it the most relevant for vocaublary analysis. URL, metadata and title could be misleading.

I included a fairly long regex code for dates in order to find most dates in the set. I did not find all dates, but I did find a lot.

I used whitespace_regex to replace multiple whitespace characters with a single space.

[34]:
```
import pandas as pd

data = pd.read_csv("news_sample.csv") # Load the data
data['content']
```

```
# Compile date-regex
date_regex = re.compile(r"""
\b(
    (?:\d{1,2}[./-]\d{1,2}[./-]\d{2,4}) | # 01/01/2025, 1.1.2025, 1-1-25
    (?:\d{4}[./-]\d{1,2}[./-]\d{1,2}) | # 2025/01/01, 2025-1-1
    (?:\b(?:\d{1,2})(?:st|nd|rd|th)?\s+(?:of\s+)?[A-Za-z]+\s+\d{4}\b) |  # 1st␣
↪January 2025, 2nd of February 2024
    (?:\b[A-Za-z]+\s+(?:\d{1,2})(?:st|nd|rd|th)?,?\s+\d{4}\b) | # January 1st␣
↪2025, March 2nd, 2023
    (?:\b[A-Za-z]+\s+\d{4}\b) # January 2025, March 2024
)\b
""", re.VERBOSE | re.IGNORECASE)
number_regex = re.compile(r'(\d+)') # "\d+" = One or more digits
email_regex = re.compile(r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b')
url_regex = re.compile(r'\b(?:http[s]?://|www\.)[^\s<>"]+|www\.[^\s<>"]+\b')
whitespace_regex = re.compile(r'\s+')

def clean_text(text):
    text = str(text).lower()
    text = re.sub(whitespace_regex, ' ', text) # Replace one or more whitespace␣
↪characters with a single space
    text = re.sub(date_regex, "<DATE>", text)
    text = re.sub(url_regex, "<URL>", text)
    text = re.sub(email_regex, "<EMAIL>", text)
    text = re.sub(number_regex, "<NUM>", text)
    return text

# for line in data['content']:
#     cleaned_line = clean_text((line))
#     print(cleaned_line)
```

2: Since I couldn't find a "clean-text" method to filter dates, I reused the regex code from above. Besides this, I simply set the relevant methods in the "clean" function to TRUE (including replacements) and thus cleaned the text.

[35]:
```
from cleantext import clean

# Replace dates with <DATE>
data['date_replaced'] = data['content'].apply(lambda x: re.sub(date_regex,␣
↪'<DATE>', x))

# Use clean-text
data['cleaned_content'] = data['date_replaced'].apply(lambda x: clean(x,
    fix_unicode=False,
    to_ascii=False,
    lower=True,
    no_line_breaks=True,
    no_urls=True,
    no_emails=True,
    no_phone_numbers=False,
    no_numbers=True,
    no_digits=True,
    no_currency_symbols=False,
    no_punct=False,
    replace_with_url="<URL>",
    replace_with_email="<EMAIL>",
    replace_with_phone_number="<PHONE>",
    replace_with_number="<NUM>",
    replace_with_digit="<NUM>",
    replace_with_currency_symbol="<CUR>",
    lang="en"
))

# Print result
# for line in data['cleaned_content']:
#     print(line)
```

3 Part 3

For unqie vocabulary before and after processing: I started by concatenating the "content" column into a single string, and then divided it into individual words using split(). Then, I converted it into a set of unique elements and calculated the length of this set.

I found the following:

Vocabulary before processing (number of unique words): 30005

Vocabulary after processing: (number of unique words): 25686

The unique vocabulary is reduced by approximately 4.300 after processing. This is equal to a reduction rate of around 14.39 %.

In order to find word frequency, I inserted every word into a dictionary containing the word as key and its appearance as value (+ 1 for every appearance in the for loop). To sort word frequency, I used the "sorted" method and a lambda function that ordered words by frequency in descending order.

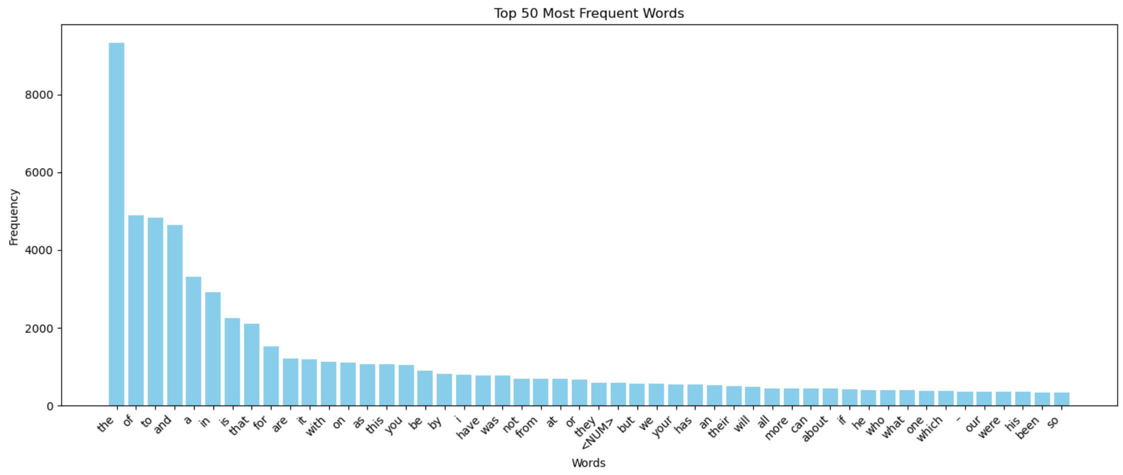
Finally, I plotted the top 50 most frequent words. I found that a lot of these words were simple stop words like "the", "of" and "to", which would have been sorted out in a more rigorous analysis.

However, if we would look from around frequency 150 and below, we see a number of interesting words such as "president", "bitcoin", "stocks", "trump", "global", "obama", "free", "research", etc.

```
[36]: import matplotlib.pyplot as plt

      # Total vocabulary size before processing
```

```python
total_vocab_before = ' '.join(data['content']).split()

# Total unique vocabulary size before processing
unique_vocab_before = len(set(total_vocab_before))

# Clean data: apply the clean_text function
data['cleaned_content'] = data['content'].apply(clean_text)

# Split into separate words and calculate the total vocabulary
total_vocab_after = ' '.join(data['cleaned_content']).split()

# Calculate size of unique vocabulary after cleaning
unique_vocab_after = len(set(total_vocab_after))

print(f"Vocabulary before processing (number of unique words):␣
 ↪{unique_vocab_before}")
print(f"Vocabulary after processing: (number of unique words):␣
 ↪{unique_vocab_after}")
print(f"Number of words removed: {unique_vocab_before - unique_vocab_after}")
print(f"Reduction rate: {(unique_vocab_before - unique_vocab_after) /␣
 ↪unique_vocab_before * 100:.2f} %")

# Calculate word frequency
def word_frequency(data):
    word_freq = {}
    for word in data:
        word_freq[word] = word_freq.get(word, 0) + 1
    return word_freq

# Sort the vocabulary by frequency
def freq_sort(data):
    return dict(sorted(data.items(), key=lambda x: x[1], reverse=True))

sorted_top_vocab = freq_sort(word_frequency(total_vocab_after))

# print(sorted_top_vocab) # Print the sorted vocabulary

def plot_top_words(data):
    # Get the 50 most common words
    top_50 = list(sorted_top_vocab.items())[:50]

    # Separate words and frequencies
    words, frequencies = zip(*top_50)

    # Plot the bar chart
    plt.figure(figsize=(14, 6))
    plt.bar(words, frequencies, color='skyblue')
    plt.xlabel('Words')
    plt.ylabel('Frequency')
    plt.title('Top 50 Most Frequent Words')
    plt.xticks(rotation=45, ha='right')
    plt.tight_layout()
    plt.show()

plot_top_words(sorted_top_vocab)
```

```
Vocabulary before processing (number of unique words): 30005
Vocabulary after processing: (number of unique words): 25686
Number of words removed: 4319
Reduction rate: 14.39 %
```


Top 50 Most Frequent Words
Frequency
Words
the of to and a in is that for are it with on as this you be by i have was not from at or they <NUM> but we your has an their will all more can about if he who what one which - our were his been so